

# ACM321 Java Lab-8


## Interfaces

There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, interfaces are such contracts.

In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

## Defining an Interface

An interface declaration consists of modifiers, the keyword `interface`, the interface name, a comma-separated list of parent interfaces (if any), and the interface body. For example:

```
 public interface GroupedInterface extends Interface1, Interface2,  
Interface3 {  
    //codes...  
}
```

## Interface Example → Animal

```
//Animal.java Interface  
  
public interface Animal {  
  
    public void sound();  
    public void move();  
    public void sleep();  
}
```

```
}
```

```
//Dog.java Class

public class Dog implements Animal {
    public String name;

    public void sound() {
        System.out.println("Barking");
    }

    public void move() {
        System.out.println("Moving");
    }

    public void sleep() {
        System.out.println("Sleeping");
    }

    //the method does not included by interface
    public void bite() {
        System.out.println("Bite someone");
    }
}
```

```
//Cat.java Class

public class Cat implements Animal{
    public void sound() {
        System.out.println("Meow");
    };

    public void move() {
        System.out.println("Cat is moving");
    };

    public void sleep() {
        System.out.println("Cat is sleeping");
    };

    public void scratch() {
        System.out.println("Cat is scratching");
    }
}
```

```
//ProgramMain.java

public class ProgramMain {

    public static void main(String[] args) {
        Dog dog1 = new Dog();

        dog1.sound();
        dog1.move();
        dog1.bite();
        dog1.sleep();
        dog1.name = "dog name";
        System.out.println(dog1.name);
    }
}
```



### Extra Notes About The Interface Body

The **interface** body can contain **abstract methods**, **default methods**, and **static methods**. →An abstract method within an interface is followed by a semicolon, but no braces (an abstract method does not contain an implementation).

→Default methods are defined with the default modifier, and

→ static methods with the static keyword.

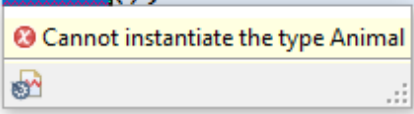
All abstract, default, and static methods in an interface are implicitly **public**, so you can **omit** the **public** modifier.

## Abstract Classes

We cannot use abstract class as regular class. The abstract class helps make regular classes is that you use there.

For example try to create a class like below;

```
1 abstract class Animal {
2     void speak() {
3         System.out.println("Animal speaking");
4     };
5 }
6
7
8 public class AbstractTry {
9
10     public static void main(String[] args) {
11         //try to call abstract animal class as regular class
12         Animal a1 = new Animal();
13         a1.speak();
14
15     }
16
17 }
18
```



```
//AbstractTry.java -> this will produce an cannot instantiate error
abstract class Animal {
    void speak() {
        System.out.println("Animal speaking");
    };
}

public class AbstractTry {

    public static void main(String[] args) {
        //try to call abstract animal class as regular class
        Animal a1 = new Animal();
        a1.speak();

    }
}
```

Lets try to create a regular Dog class and extend it from Animal abstract class.

```

1  abstract class Animal {
2      void speak() {
3          System.out.println("Animal speaking");
4      };
5  }
6
7  class Dog extends Animal {
8      public String name;
9  }
10
11 public class AbstractTry {
12
13     public static void main(String[] args) {
14         //try to call abstract animal class as regular class
15         Dog d1 = new Dog();
16         d1.speak();
17     }
18 }
19
20 }
21

```

```

//AbstractTry.java

abstract class Animal {
    void speak() {
        System.out.println("Animal speaking");
    };
}

class Dog extends Animal {
    public String name;
}

//you can also create <class Cat extends Animal> here, like Dog class.

public class AbstractTry {

    public static void main(String[] args) {
        //try to call abstract animal class as regular class
        Dog d1 = new Dog();
        d1.speak();
    }

}

```

## Example of Abstract Classes and Abstract Methods together

The abstract methods `speak()` and `eat()` below, should be implemented on subclasses.

The abstract class helps us to organize what a subclass should have. It simply says; every single `Animal` must have these methods.

```
//Animal.java abstract class

public abstract class Animal {
    public void move() {
        System.out.println("Moving");
    }

    public void sleep() {
        System.out.println("Sleeping");
    }

    //abstract method is a method that is not implemented yet.
    //implementing means the we put some code between the brackets {}.
    //in abstract methods we are not going to implement these methods.
    abstract void speak();
    abstract void eat();
}
```

```
//Dog.java class

public class Dog extends Animal {

    public void speak() {
        System.out.println("Dog is barking");
    }

    public void eat() {
        System.out.println("Dog is eating meat");
    }

    public void bite() {
        System.out.println("Dog is biting someone! ");
    }
}
```

```
//Cat.java class

public class Cat extends Animal {
    public void speak() {
        System.out.println("Cat is meowing");
    }
}
```

```
public void eat() {  
    System.out.println("Cat is drinking milk");  
}  
  
public void scratch() {  
    System.out.println("Cat is scracathing something");  
}  
}
```